

Tests

SI539 - Chapter 6 Lenz

Textbook: Build Your own Ruby on Rails Application by Patrick Lenz (ISBN:978-0-975-8419-5-2)

Automated Testing

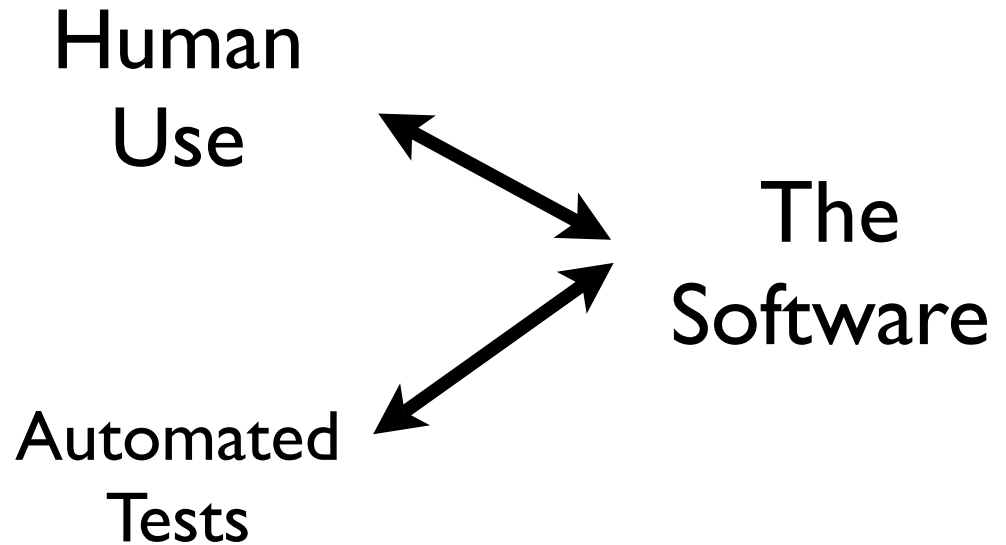
- Building automated tests is an important part of disciplined software development
- Rails has amazingly good support for building tests

Why Automated Testing

- As systems get increasingly complex, having a human test all the possibilities becomes nearly impossible and certainly very costly
- Automated testing puts the computer to work testing
- You can make a change to code, run the tests, and be pretty sure that things will work if the tests pass

The Basic Diagram

Tests effective prod and poke the software much like a human tester would do - but often with even more obsessive attention to detail.



Tests in the Development Cycle

- In many organizations there is a policy of “write a test first”
- When you want a new feature
 - Write a unit test that exercises the feature - of course it will initially fail
 - Capture the specifications and requirements in the test
 - Then work on the software until it passes the test

Tests and Quality Assurance

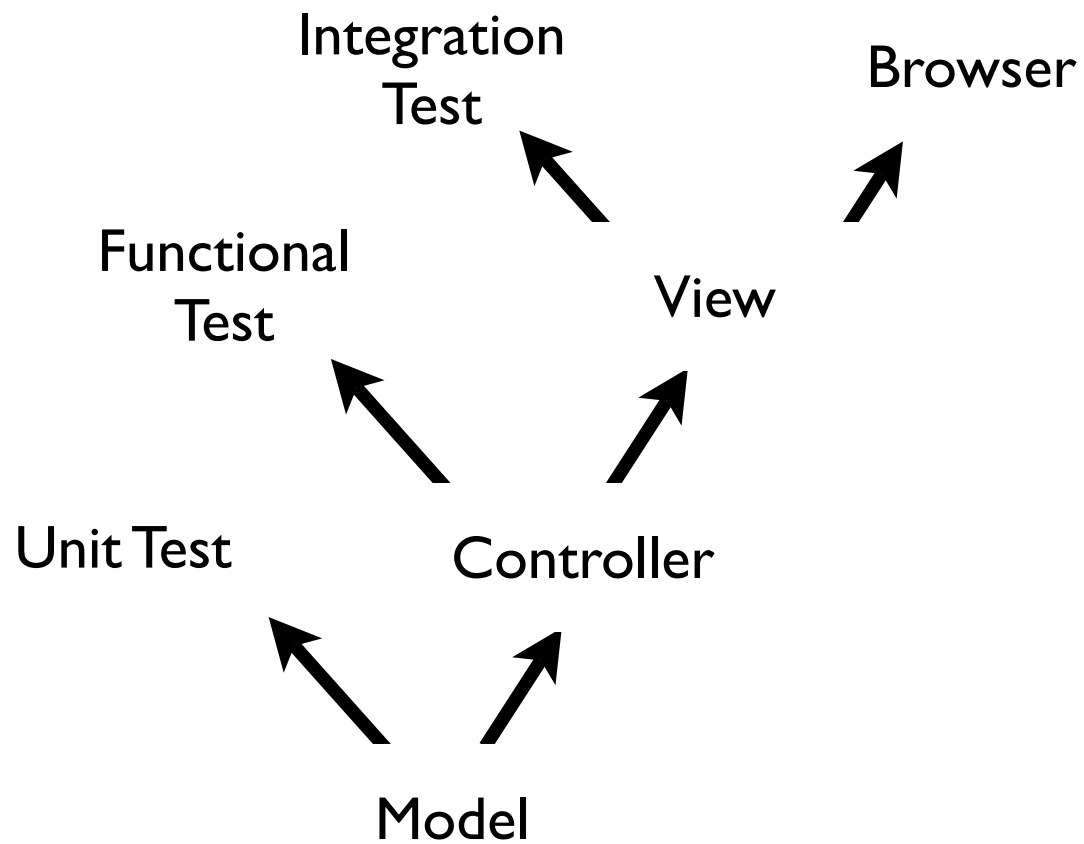
- Some Organizations insist that when human testing reveals a bug - a new test must be created to reproduce the bug before the bug is fixed
- The test is added to the test suite forever
- Insures that bugs do not reappear

To Test or Not to Test...

- The amount of energy invested in building tests will depend on the project
- Scenario: You are a high-priced consultant writing software that you will hand to someone else to maintain and run - you are paid very well but they get to modify the code
- Scenario: You are a multi-person team building inter-related modules that need to interact but people are completing their modules at different times

Three levels of tests

- Unit Testing of the Models
- Functional Testing of the Controllers
- Integration testing of the overall Application across multiple controllers (not covered)



Unit Tests for Models

Outline

- You populate a test database
- You can start by looking at controller code and pulling it out into your unit test
- Build your unit test for a model one test at a time - test repeatedly as you build

Unit Test Database Environment

- Unlike development - the test database is created fresh and new for each unit test
- Unit tests can save data into the database but it vanishes at the end of each unit test

Fixtures

- Since the database is made fresh for each unit test we must fill it with data
- Fixtures are written in YAML - Yet Another Markup Language
 - test/fixtures
- Fixtures poke data directly into the tables including the id field
- Fixtures work with the test database - not the development database

<http://ar.rubyonrails.org/classes/Fixtures.html>

test/fixtures/members.yml

Read about fixtures at <http://ar.rubyonrails.org/classes/Fixtures.html>
one:

id: 1
name: Dr. Chuck
email: csev@umich.edu

two:
id: 2
name: The Sakaiger
email: ger@sakaiger.com

Two records. Note that we specify the value for the id column. This allows us to link data between two tables using “foreign keys”.

Making the Test Database Schema

- Similar to rake db:migrate - but for the test database
- Copies schema from development to test - not the data
- Does not run the fixtures - unit tests run the fixtures
- rake db:test:prepare

```
charles-severances-macbook-pro:trunk csev$ rake db:test:prepare  
(in /Users/csev/dev/toozday/trunk)  
charles-severances-macbook-pro:trunk csev$ ls -l db/test.sqlite3  
-rw-r--r--  1 csev  staff  12288 Nov 21 04:55 db/test.sqlite3  
charles-severances-macbook-pro:trunk csev$
```

If you forget db:test:prepare...

```
charles-severances-macbook-pro:trunk csev$ ruby test/unit/comment_test.rb
Loaded suite test/unit/comment_test
Started
EE
Finished in 0.046614 seconds.

1) Error:
test_truth(CommentTest):
ActiveRecord::StatementInvalid: SQLite3::SQLException: no such table: sitetypes:
DELETE FROM sitetypes WHERE 1=1
    /System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/gems/1.8
/gems/activerecord-1.15.3/lib/active_record/connection_adapters/abstract_adapter
.rb:128:in `log'
    /System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/gems/1.8
/gems/activerecord-1.15.3/lib/active_record/connection_adapters/sqlite_adapter.r
```


Your First Unit Test

- When you made your model, rails generates which contain skeleton unit tests that always pass

tests/unit/member_test.rb

```
require File.dirname(__FILE__) + '/../test_helper'

class MemberTest < Test::Unit::TestCase
  fixtures :members ←
  def test_truth
    assert_true ←
  end
end
```

Running the Unit Test

- Loads the fixtures - runs the test which passes because it simply asserts true
- You can see the fixtures if you open the database in the browser

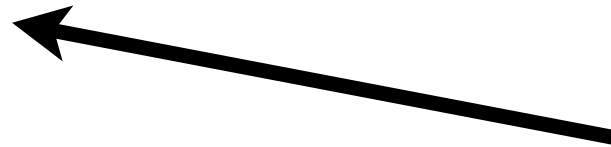
```
charles-severances-macbook-air:asn8 csev$ ruby test/unit/member_test.rb
Loaded suite test/unit/member_test
Started
.
Finished in 0.080376 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
charles-severances-macbook-air:asn8 csev$
```

Making a non-trivial Unit Test

- Controller code is a good place to find model test code

```
def members  
  @members = Member.find(:all)  
  logger.info @members  
end
```



It looks for methods of any name with a prefix of “test”.

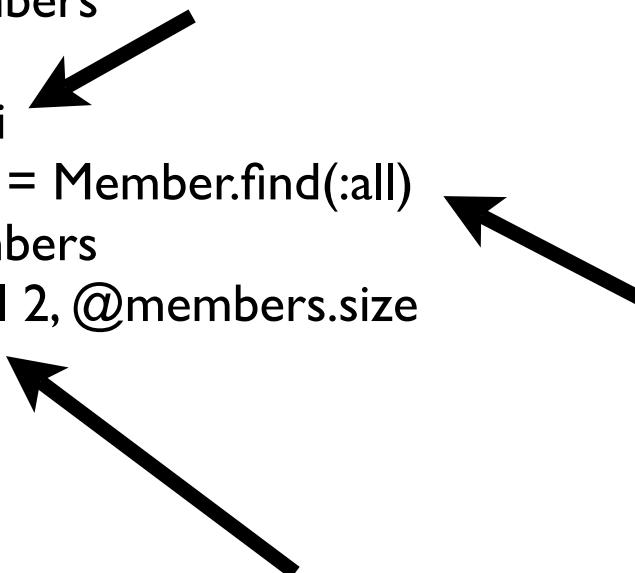
Do something and then make sure results are as expected.

You can debug print - but we remove this after testing.

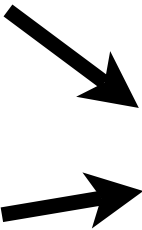
```
require File.dirname(__FILE__) + '/../test_helper'

class MemberTest < Test::Unit::TestCase
  fixtures :members

  def test_multi
    @members = Member.find(:all)
    puts @members
    assert_equal 2, @members.size
  end
end
```



Running our Unit Test



```
charles-severances-macbook-air:asn8 csev$ ruby test/unit/member_test.rb
Loaded suite test/unit/member_test
Started
#<Member:0x2257360>
#<Member:0x2257324>
.
Finished in 0.080362 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
charles-severances-macbook-air:asn8 csev$
```

The image shows a terminal window with the output of a Ruby unit test. Two black arrows point to the output lines: the first arrow points to the line "#<Member:0x2257360>" and the second arrow points to the line "#<Member:0x2257324>".

```
def test_multi
  @members = Member.find(:all)
  puts @members
  assert_equal 3, @members.size
end
```

We are testing the
fixtures and the model
code.

one:
id: 1
name: Dr. Chuck
email: csev@umich.edu

two:
id: 2
name: The Sakaiger
email: ger@sakaiger.com

```
charles-severances-macbook-air:asn8 csev$ ruby test/unit/member_test.rb
Loaded suite test/unit/member_test
Started
#<Member:0x2257360>
#<Member:0x2257324>
F
Finished in 0.108057 seconds.

1) Failure:
test_multi(MemberTest) [test/unit/member_test.rb:9]:
<3> expected but was
<2>.

1 tests, 1 assertions, 1 failures, 0 errors
charles-severances-macbook-air:asn8 csev$
```

For coverage

- Look through the controller code and see how the controller is using the model - pull that out into unit tests
- You always want lots of unit tests even if they seem a bit trivial
- You are just making sure the basic stuff really works

Scan controller for things to test

```
def thanks
  memb = Member.create()
  memb.name = params[:yourname]
  memb.email = params[:yourmail]
  memb.save
  @barcelona = memb.id
end
```

```
def delete
  memb = Member.find(params[:id])
  memb.destroy()
  flash[:notice] = "User Deleted"
  redirect_to :action => 'members'
end
```

```
def test_insert
  memb = Member.create()
  memb.name = "Wally"
  memb.email = "wally@sakaiger.com"
  assert memb.save
  @members = Member.find(:all)
  # puts @members
  assert_equal 3, @members.size
end
```

```
def test_destroy
  memb = Member.find(1)
  assert memb.destroy()
  @members = Member.find(:all)
  # puts @members
  assert_equal 1, @members.size
end
```


Smooth Unit Test Run

```
charles-severances-macbook-air:assn8 csev$ ruby test/unit/member_test.rb
Loaded suite test/unit/member_test
Started
...
Finished in 0.089325 seconds.

3 tests, 5 assertions, 0 failures, 0 errors
charles-severances-macbook-air:assn8 csev$
```

Unit Test Summary

- Tests Models
 - Build Fixtures
- Look in the controller and see how the controller(s) use the model and extract that code
- Assert the obvious - even if they seem trivial - they are “free”

Unit Test Assertions

- `assert(boolean,message)` - Fails if boolean is false
- `assert_equal(expected, actual, message)`
- `assert_not_equal(expected, actual, message)`
- `assert_nil(object, message)`
- `assert_not_nil(object, message)`

<http://www.ruby-doc.org/stdlib/libdoc/test/unit/rdoc/classes/Test/Unit/Assertions.html>

Unit Test Assertions

- `assert_match(pattern, string, message)`
- `assert_no_match(pattern, string, message)`
- `assert_value(activerecord_object)`
- `flunk(message)` - Just plain fails

<http://www.ruby-doc.org/stdlib/libdoc/test/unit/rdoc/classes/Test/Unit/Assertions.html>

Functional Testing Controllers

Functional Testing

- Emulates a browser talking to a controller
 - Fake HTTP Request
 - Fake HTTP Response
 - Fake Post Data
 - Fake Session


tests/functional/
one_controller.rb

Functional test
skeleton created by
ruby script/generate
controller.


```
require File.dirname(__FILE__) + '/../test_helper'  
require 'one_controller'
```

```
# Re-raise errors caught by the controller.  
class OneController; def rescue_action(e) raise e end; end
```

```
class OneControllerTest < Test::Unit::TestCase  
  def setup  
    @controller = OneController.new  
    @request     = ActionController::TestRequest.new  
    @response    = ActionController::TestResponse.new  
  end
```



```
# Replace this with your real tests.  
def test_true  
  assert true  
end  
end
```




Running A Functional Test

```
charles-severances-macbook-air:asn8 csev$ ruby test/functional/one_controller_test.rb
Loaded suite test/functional/one_controller_test
Started
.
Finished in 0.065089 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
charles-severances-macbook-air:asn8 csev$
```


In functional tests - you “simulate” a browser doing the request - response cycle.

```
def test_one  
  get :index  
  assert_response :success  
end
```



Get the page for the index action.

Assert that we got a normal HTTP Response (aka Success).

```
def test_index
  get :index
  assert_select 'a.selected', 'About'
end

def test_join
  get :join
  assert_select 'a.selected', 'Application'
end
```

You can also “peer into” the returned HTML and check to see if things are working.

Look for the <a> tag with the class selected and check the text in that tag and compare to “Application”

```
<li><a href="/one">About</a></li>
<li><a href="/one/contact">Contact</a></li>
<li><a href="/one/pictures">Pictures</a></li>
<li><a href="/one/members">Membership</a></li>
<li><a href="/one/join" class="selected" >Application</a></li>
```

```
def test_join_success
  post :thanks, 'yourname' => 'Chuck',
    'yourmail' => 'csev@umich.edu'
  assert_nil flash[:notice]
  assert_response :success
  assert_template 'thanks'
end
```

You can also do a post, passing form parameters. You can check the flash to make sure there were no errors, assert that we got a success return code in the HTML response, and make sure that we were sent back the thanks template.

```
def test_join_error_01
  post :thanks
  assert_not_nil flash[:notice]
  assert_redirected_to :action => 'join'
end
```

```
def test_join_error_02
  post :thanks, 'yourname' => 'Chuck'
  assert_not_nil flash[:notice]
  assert_redirected_to :action => 'join'
end
```

We can test for one or both form parameters missing. We can insist that we get an error message and instead of a “success” HTML response code and we can check to make sure we are redirected to the ‘join’ action.

Successful Run

```
charles-severances-macbook-air:asn8 csev$ ruby test/functional/one_controller_test.rb
Loaded suite test/functional/one_controller_test
Started
.....
Finished in 0.119781 seconds.

6 tests, 10 assertions, 0 failures, 0 errors
charles-severances-macbook-air:asn8 csev$
```

Functional Test Assertions

- `assert_response(type, message)` :success, :redirect, :missing, :error
- `assert_redirected_to(options, message)` - options is like `url_for`
- `assert_template(expected, message)` - example 'portal/index'
- `assert_select` needs a cheat sheet
- This is best done looking at sample tests in books, etc.

http://labnotes.org/svn/public/ruby/rails_plugins/assert_select/cheat/assert_select.pdf

Running Tests Using rake

- `rake test` # Test all units and functionals
- `rake test:functionals` # Run the functional tests in test/functional
- `rake test:integration` # Run the integration tests in test/integration
- `rake test:plugins` # Run the plugin tests in vendor/plugins/**/
- `rake test:recent` # Test recent changes
- `rake test:units` # Run the unit tests in test/unit

```
charles-severances-macbook-air:assn8 csev$ rake test
(in /Users/csev/Desktop/teach/a539/w08/rails_apps/assn8)
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby -Ilib:test "/System/Library/Frameworks/Ruby
.framework/Versions/1.8/usr/lib/ruby/gems/1.8/gems/rake-0.7.3/lib/rake/rake_test_loader.rb" "test/unit/member_t
est.rb"
Loaded suite /System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/gems/1.8/gems/rake-0.7.3/lib/r
ake/rake_test_loader
Started
...
Finished in 0.090261 seconds.

3 tests, 5 assertions, 0 failures, 0 errors
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby -Ilib:test "/System/Library/Frameworks/Ruby
.framework/Versions/1.8/usr/lib/ruby/gems/1.8/gems/rake-0.7.3/lib/rake/rake_test_loader.rb" "test/functional/on
e_controller_test.rb"
Loaded suite /System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/gems/1.8/gems/rake-0.7.3/lib/r
ake/rake_test_loader
Started
.....
Finished in 0.218506 seconds.

6 tests, 10 assertions, 0 failures, 0 errors
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby -Ilib:test "/System/Library/Frameworks/Ruby
.framework/Versions/1.8/usr/lib/ruby/gems/1.8/gems/rake-0.7.3/lib/rake/rake_test_loader.rb"
charles-severances-macbook-air:assn8 csev$
```


Functional Testing Summary

- Simulated browser talking to one controller
- Goal - test coverage of all code paths - some tools monitor this

Summary

- Unit tests can be a powerful developer tool
- Unit tests insure against regressions - particularly over time or across multi-person teams
- Initially you will likely debug first and then write the tests later
- Over time you may move to writing the tests first and then writing the application code after the tests